CSC 311 – Winter 2022
Design and Analysis of Algorithms
2. Growth of functions and asymptotic notation

Prof. Mohamed Menai
Department of Computer Science
King Saud University

Outline

- Asymptotic notation
- The O-notation
- The Ω-notation
- The Θ-notation
- The o-notation
- The ω-notation

Overview

- Order of growth of functions provides a simple characterization of efficiency
- Allows for comparison of relative performance between alternative algorithms
- Concerned with ***asymptotic*** efficiency of algorithms
- Best asymptotic efficiency usually is best choice except for smaller inputs
- Several standard methods to simplify asymptotic analysis of algorithms

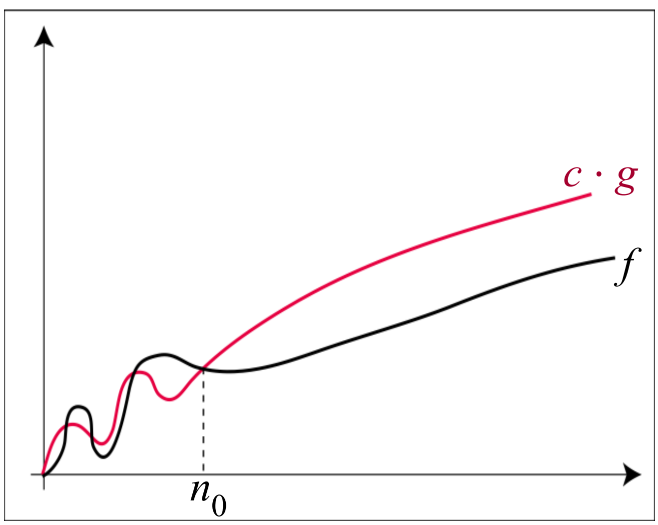
Asymptotic Notation

- Applies to functions whose domains are the set of natural numbers $N = \{0,1,2,\ldots\}$
- If time resource $T(n)$ is being analyzed, the function's range is usually the set of non-negative real numbers: $T(n) \in R^+$
- If space resource $S(n)$ is being analyzed, the function's range is usually also the set of natural numbers: $S(n) \in N$

The O-Notation

- The O-notation is an asymptotic upper bound.
- $f(n) = O(g(n))$ pronounced "f of n is big-oh of g of n":

$$\boldsymbol{O(g(n)) = \{ f(n) : \exists c > 0, n_0 > 0 \text{ so that } \forall n \geq n_0 : 0 \leq f(n) \leq c \cdot g(n)\}}$$



5

Using the Definition of the *O*-Notation

Example: Show that $f(x) = x^2 + 2x + 1$ is $O(x^2)$
Solution: Since when $x > 1$, $x < x^2$ and $1 < x^2$

$$0 \leq x^2 + 2x + 1 \leq x^2 + 2x^2 + x^2 = 4x^2$$
$$f(x) \ \text{ is } O(x^2)$$

– Can take $c = 4$ and $n_0 = 1$ as witnesses to show that

Combinations of Functions

- If $f_1(x)$ is $O(g_1(x))$ and $f_2(x)$ is $O(g_2(x))$ then
  $(f_1 + f_2)(x)$ is $O(\max(g_1(x), g_2(x)))$.
- If $f_1(x)$ and $f_2(x)$ are both $O(g(x))$ then
  $(f_1 + f_2)(x)$ is $O(g(x))$.
- If $f_1(x)$ is $O(g_1(x))$ and $f_2(x)$ is $O(g_2(x))$ then
  $(f_1 f_2)(x)$ is $O(g_1(x) g_2(x))$.
- $f(x) = O(g(x)) \Rightarrow f(x) + g(x) = O(g(x))$

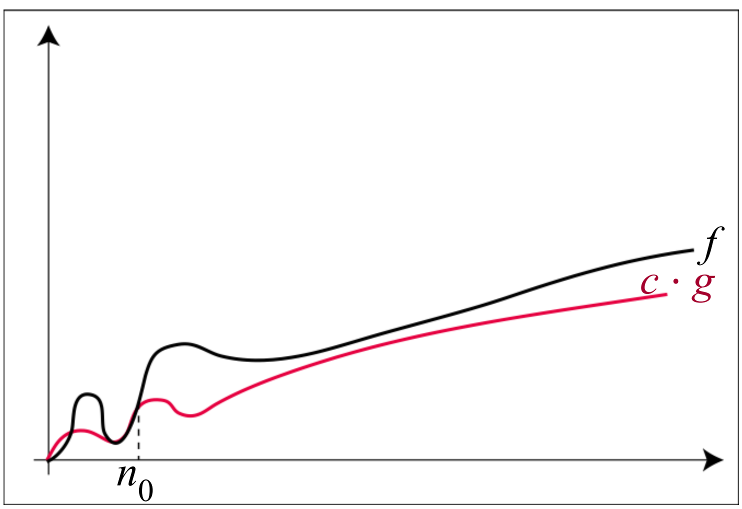
The Ω-Notation

- The O-notation provides an asymptotic upper bound on a function.
- The Ω-notation provides an asymptotic lower bound on a function.
- $f(n) = \Omega(g(n))$ pronounced "f of n is big-omega of g of n":

$$\Omega(g(n)) = \{ f(n) : \exists c > 0, n_0 > 0 \text{ so that } \forall n \geq n_0: f(n) \geq c \cdot g(n) \geq 0\}$$

The Ω-Notation

$\Omega(g(n)) = \{ f(n) : \exists c > 0, n_0 > 0$ **so that**
$\forall n \geq n_0 : f(n) \geq c \cdot g(n) \geq 0\}$



9

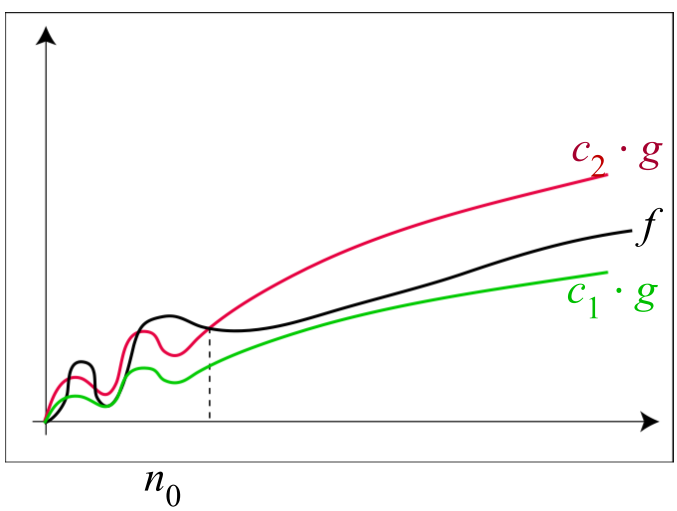
The Ω-Notation

Example: Show that $f(x) = 8x^3 + 5x^2 + 7$ is $\Omega(g(x))$ where $g(x) = x^3$

Solution: $f(x) = 8x^3 + 5x^2 + 7 \geq 8x^3$ for all positive real numbers *x*.

The Θ-Notation

- The Θ-notation is an asymptotically tight bound on $f(n)$.
- Θ-notation is a stronger notion than O-notation.
  $\Theta(g(n))$ is a sub-set of $O(g(n))$
- $g(n)$ asymptotically bounds a function from above and below.



11

The Θ-Notation

- $\Theta(g(n))$ is the set of functions:
  $\boldsymbol{\Theta(g(n)) = \{ f(n) : \exists c_1, c_2 > 0, n_0 > 0}$ **so that**
  $\boldsymbol{\forall n \geq n_0 : c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)\}}$
- A function $f(n)$ belongs to the set $\Theta(g(n))$ if there exist positive constants c_1 and c_2 such that it can be "sandwiched" between $c_1 \cdot g(n)$ and $c_2 \cdot g(n)$, for sufficiently large n.
- Notation: $f(n) = \Theta(g(n))$

Theorem

For any two functions $f(n)$ and $g(n)$, we have $f(n) = \Theta(g(n))$ if and only if $f(n) = O(g(n))$ and $f(n) = \Omega\ (g(n))$.

The Θ-Notation Estimates for Polynomials

Theorem: Let $f(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_o$ where $a_0, a_1, \ldots, a_n$ are real numbers with $a_n \neq 0$.
Then $f(x)$ is of order x^n (or $\Theta(x^n)$).

Example:
The polynomial $f(x) = 8x^5 + 5x^2 + 10$ is order of x^5 (or $\Theta(x^5)$).
The polynomial $f(x) = 8x^{199} + 7x^{100} + x^{99} + 5x^2 + 25$ is order of x^{199} (or $\Theta(x^{199})$).

14

The o-Notation

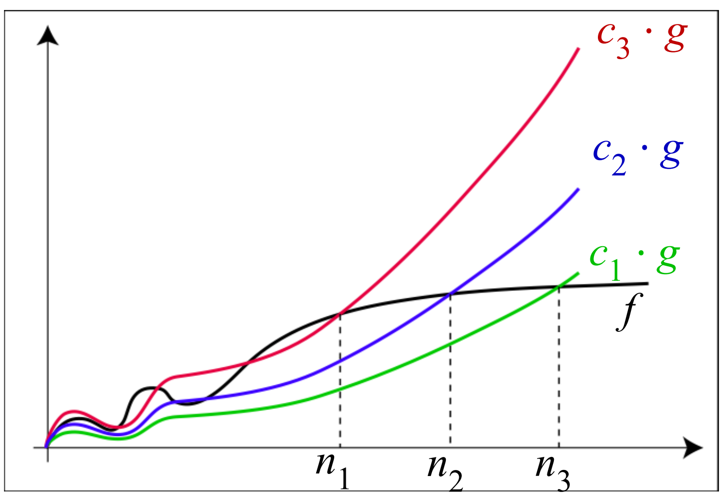
- The asymptotic upper bound provided by the O-notation may or may not be asymptotically tight:
  - The bound $2n^2 = O(n^2)$ is asymptotically tight.
  - The bound $2n = O(n^2)$ is not.
- The o-notation is used to denote an upper bound that is not asymptotically tight.

  $f(n) = o(g(n))$ pronounced "f of n is little-oh of g of n": $\boldsymbol{o(g(n)) = \{ f(n) : \forall c > 0\ \exists n_0 > 0 \text{ so that } \forall n \geq n_0 : 0 \leq f(n) \leq c \cdot g(n)\}}$
- For example, $2n = o(n^2)$, but $2n^2 \neq o(n^2)$

15

The *o*-Notation

$o(g(n)) = \{ f(n) : \forall c > 0\ \exists n_0 > 0$ **so that** $\forall n \geq n_0$: $0 \leq f(n) \leq c \cdot g(n)\}$



The o-Notation

$o(g(n)) = \{ f(n) : \forall c > 0\ \exists n_0 > 0$ so that $\forall n \geq n_0 : 0 \leq f(n) \leq c \cdot g(n)\}$

- In $f(n) = O(g(n))$, the bound $f(n) \leq c \cdot g(n)$ holds for some constant $c > 0$.
- In $f(n) = o(g(n))$, the bound $f(n) \leq c \cdot g(n)$ holds for all constants $c > 0$.
- Intuitively, the function $f(n)$ becomes insignificant relative to $g(n)$, as n approaches infinity: $$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$$

17

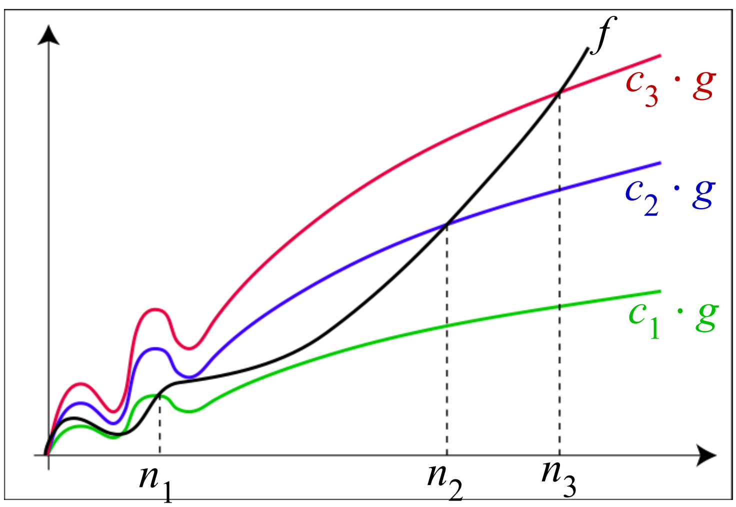
The ω-Notation

- The ω-notation is to Ω-notation, as the *o*-notation is to *O*-notation.
- The ω-notation is used to denote a lower bound that is not asymptotically tight.
- $f(n) \in \omega(g(n))$ if and only if $g(n) \in o(f(n))$
- pronounced "*f* of *n* is little-omega of *g* of *n*".
- $f(n) \in \omega(g(n))$ implies that: $\lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty$

18

The ω-Notation

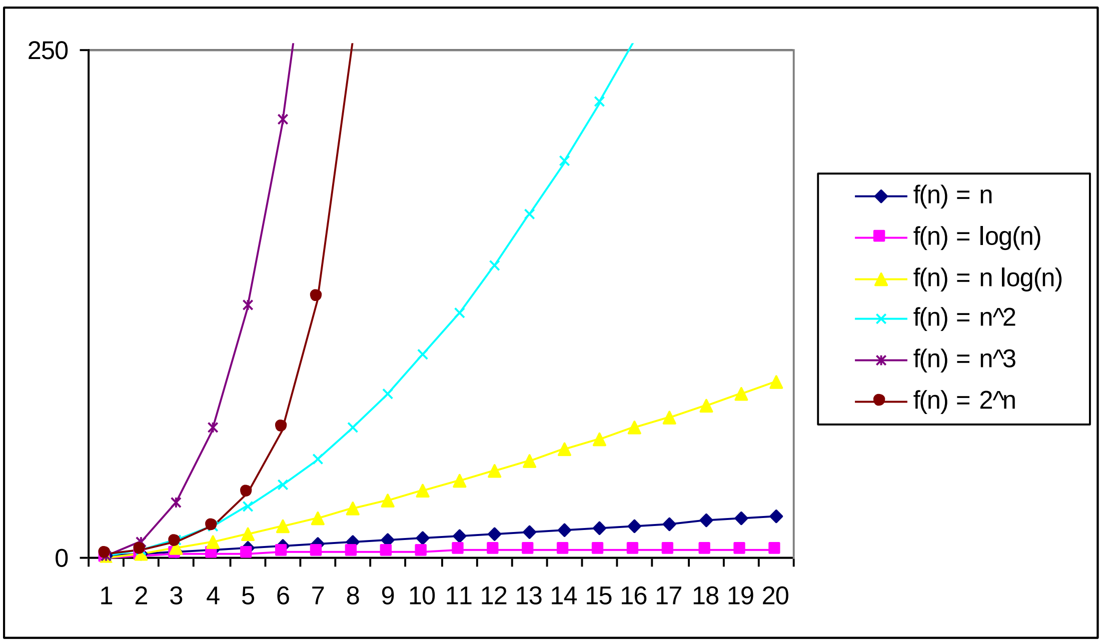
$\boldsymbol{\omega(g(n)) = \{ f(n) : \forall c > 0\ \exists n_0 > 0 \text{ so that } \forall n \geq n_0: f(n) \geq c \cdot g(n) \geq 0\}}$



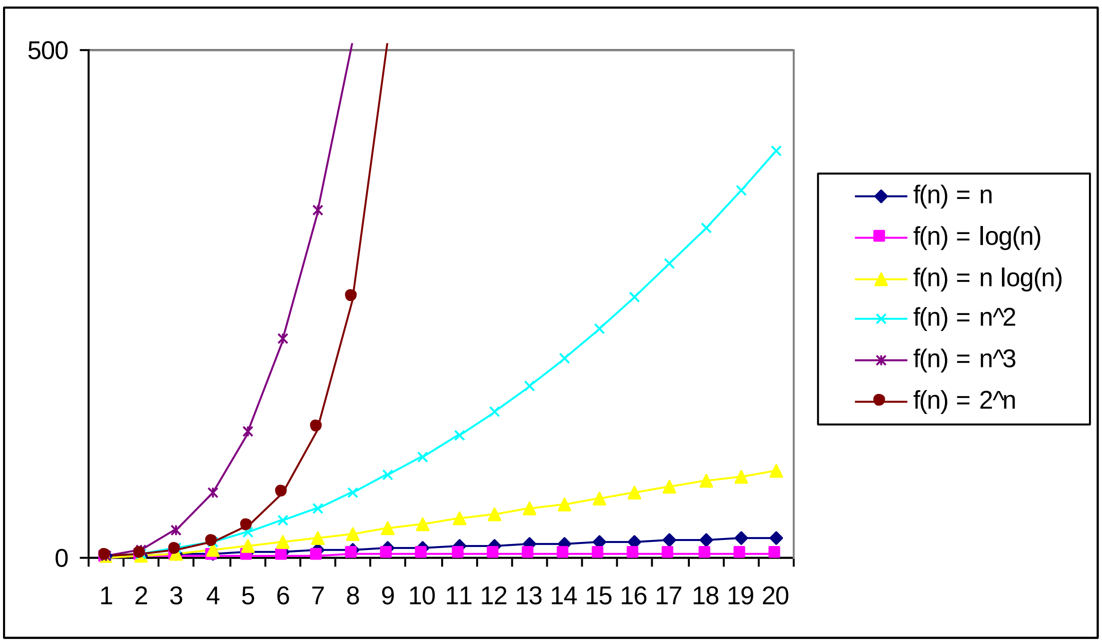
Intuition for Asymptotic Notation

- Big-Oh
  - $f(n)$ is $O(g(n))$ if $f(n)$ is asymptotically less than or equal to $g(n)$.
- Big-Omega
  - $f(n)$ is $\Omega(g(n))$ if $f(n)$ is asymptotically greater than or equal to $g(n)$.
- Theta
  - $f(n)$ is $\Theta(g(n))$ if $f(n)$ is asymptotically equal to $g(n)$.
- Little-oh
  - $f(n)$ is $o(g(n))$ if $f(n)$ is asymptotically strictly less than $g(n)$.
- Little-omega
  - $f(n)$ is $\omega(g(n))$ if $f(n)$ is asymptotically strictly greater than $g(n)$.

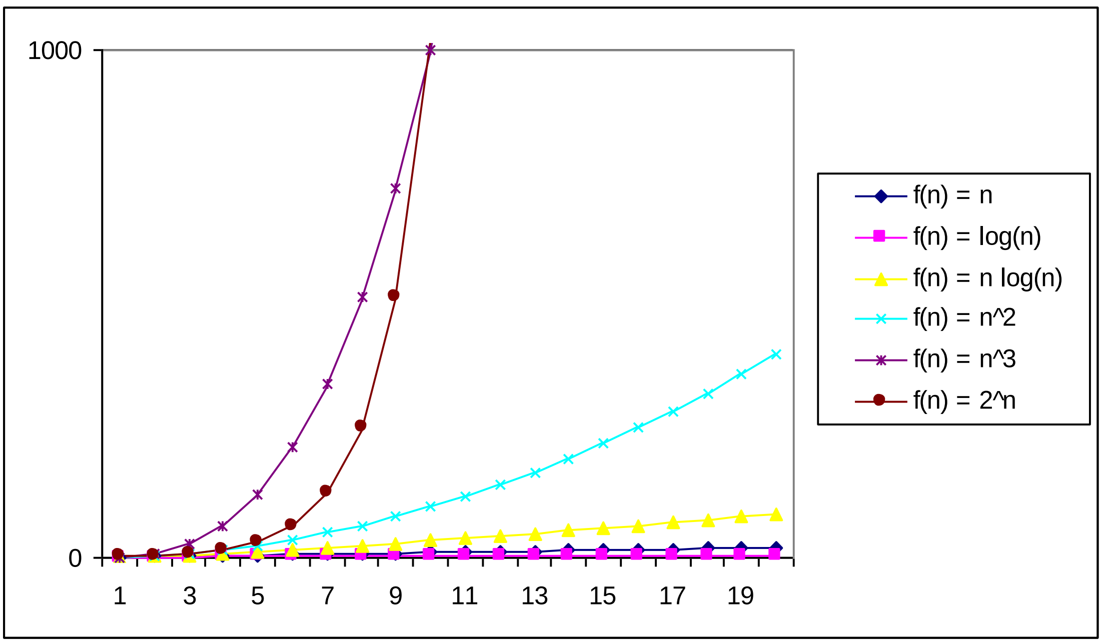
Practical Complexity



Practical Complexity

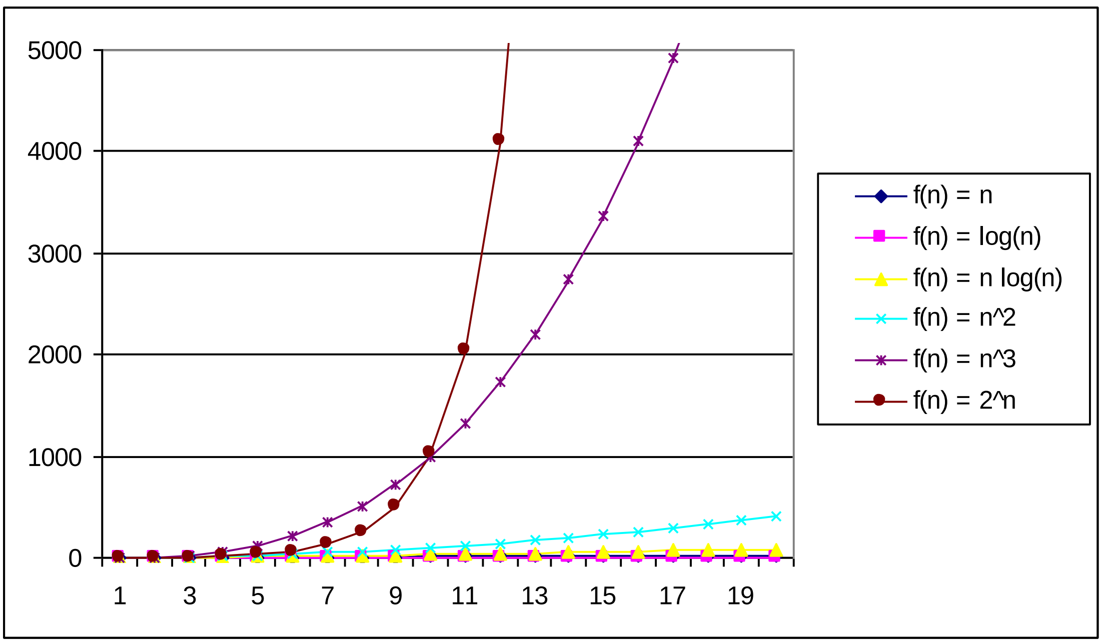


Practical Complexity



23

Practical Complexity



24

Comparison of Functions

Transitivity

- $f(n) = O(g(n))$ and $g(n) = O(h(n)) \Rightarrow f(n) = O(h(n))$
- $f(n) = \Omega(g(n))$ and $g(n) = \Omega(h(n)) \Rightarrow f(n) = \Omega(h(n))$
- $f(n) = \Theta(g(n))$ and $g(n) = \Theta(h(n)) \Rightarrow f(n) = \Theta(h(n))$

Reflexivity

- $f(n) = O(f(n))$
  $f(n) = \Omega(f(n))$
  $f(n) = \Theta(f(n))$

Comparison of Functions

Symmetry

- $f(n) = \Theta(g(n)) \Leftrightarrow g(n) = \Theta(f(n))$

Transpose Symmetry

- $f(n) = O(g(n)) \Leftrightarrow g(n) = \Omega(f(n))$
- $f(n) = o(g(n)) \Leftrightarrow g(n) = \omega(f(n))$

Asymptotic Analysis and Limits

$$\text{if } \lim_{n\to\infty}\frac{f(n)}{g(n)} = 0, \text{ then } f(n) = o(g(n)).$$

$$\text{if } \lim_{n\to\infty}\frac{f(n)}{g(n)} = c, \text{ for some constant } c > 0, \text{ then } f(n) = \Theta(g(n)).$$

$$\text{if } \lim_{n\to\infty}\frac{f(n)}{g(n)} = 0, \text{ then } a^{f(n)} = o\left(a^{g(n)}\right), \text{ for any } a > 1.$$

$$f(n) = o(g(n)) \Rightarrow a^{f(n)} = o\left(a^{g(n)}\right), \text{ for any } a > 1.$$

Standard Notation and Common Functions

- Important relationships
  - For all real constants a and b such that $a > 1$,
    $$n^b = o(a^n)$$
    that is, any exponential function with a base strictly greater than unity grows faster than any polynomial function.
  - For all real constants a and b such that $a > 0$,
    $$\log^b n = o(n^a)$$
    that is, any positive polynomial function grows faster than any polylogarithmic function.

Standard Notation and Common Functions

- Factorials
  - For all n the function $n!$ or "n factorial" is given by
    $n! = n \times (n-1) \times (n-2) \times (n-3) \times \ldots \times 2 \times 1$
  - It can be established that
    $n! = o(n^n)$
    $n! = \omega(2^n)$
    $\log(n!) = \Theta(n \log n)$

29

Asymptotic Running Time of Algorithms

- We consider algorithm *A* better than algorithm *B* if:

$$T_A(n) = o(T_B(n))$$

- Why is it acceptable to ignore the behavior of algorithms for small inputs?
- Why is it acceptable to ignore the constants?
- What do we gain by using asymptotic notation?

30

Things to Remember

- Asymptotic analysis studies how the values of functions compare as their arguments grow without bounds.
- Ignores constants and the behavior of the function for small arguments.
- Acceptable because all algorithms are fast for small inputs and growth of running time is more important than constant factors.

Things to Remember

- Ignoring the usually unimportant details, we obtain a representation that succinctly describes the growth of a function as its argument grows and thus allows us to make comparisons between algorithms in terms of their efficiency.

Reading

Chapter 2 (Sections 2.1, 2.2)

Anany Levitin, Introduction to the design and analysis of algorithms, 3rd Edition, Pearson, 2012.